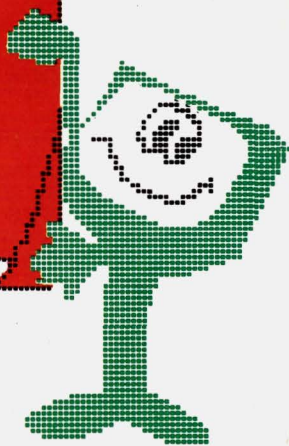
VIDEO BASIC

20 VIDEOLEZIONI DI BASIC
PER IMPARARE COL VIC 20

GRUPPO
EDITORIALE
JACKSON

CPU: organizzazione esterna e interna

I BUS

Il linguaggio macchina

Vantaggi e svantaggi dell'Assembler

SYS, USR

Memorizzazione di programmi in linguaggio macchina

Videoesercizi

Videogioco n° 18

18

COMMODORE
VIC 20

VIDEO BASIC VIC 20
Pubblicazione quattordicinale
edita dal Gruppo Editoriale Jackson
Direttore Responsabile:
Giampietro Zanga
Direttore e Coordinatore
Editoriale: Roberto Pancaldi
Autore: Softidea - Via Indipendenza 88 - Como
Redazione software:
Francesco Franceschini, Enrico Braglia,
Fabio Calanca
Segretaria di Redazione:
Marta Menegardo
Progetto grafico:
Studio Nuovaidea - Via Longhi 16 - Milano
Impaginazione:
Silvana Corbelli
Illustrazioni:
Cinzia Ferrari, Silvano Scolari
Fotografie:
Marcello Longhini
Distribuzione: SODIP
Via Zuretti, 12 - Milano
Fotocomposizione: Lineacomp S.r.l.
Via Rosellini, 12 - Milano
Stampa: Grafika '78
Via Trieste, 20 - Pioltello (MI)
Direzione e Redazione:
Via Rosellini, 12 - 20124 Milano
Tel. 02/6880951/5

Autorizzazione alla pubblicazione Tribunale di Milano n° 422 del 22-9-1984
Spedizione in abbonamento postale Gruppo II/70 (autorizzazione della Direzione Provinciale delle PPTT di Milano).
Prezzo del fascicolo L. 8.000
Abbonamento comprensivo di 5 raccoglitori L. 165.000
I versamenti vanno indirizzati a: Gruppo Editoriale Jackson S.r.l. - Via Rosellini, 12 20124 Milano, mediante emissione di assegno bancario o cartolina vaglia oppure utilizzando il c.c.p. n° 11666203.
I numeri arretrati possono essere richiesti direttamente all'editore inviando L. 10.000 cdu. mediante assegno bancario o vaglia postale o francobolli.
Non vengono effettuate spedizioni contrassegno.

Gruppo Editoriale Jackson

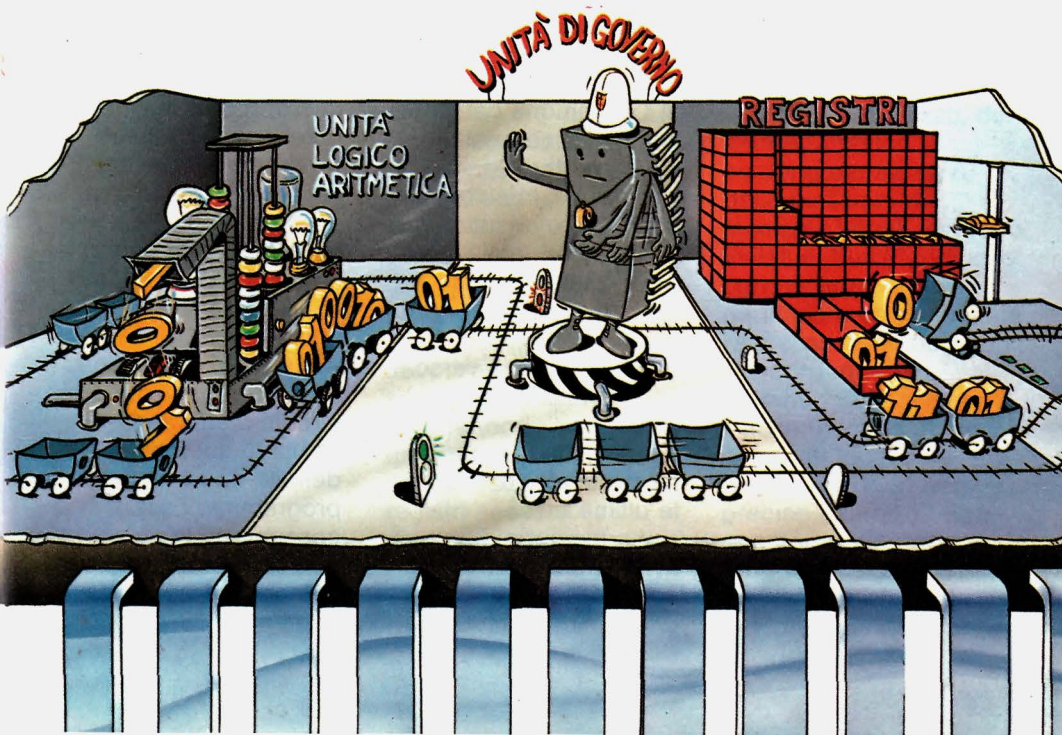
SOMMARIO

Introduzione

Prima di andare avanti ... facciamo un passo indietro. Per introdurre, infatti, il tanto famigerato linguaggio macchina (l. m. per chi vuol darsi un tono) e l'altrettanto noto assembler, occorre rinfrescare i concetti base nella CPU, la sua organizzazione interna e esterna, i bus.
Non è un'inutile esercitazione accademica; per ottenere mirabolanti prestazioni dal proprio computer è necessario - a volte indispensabile quando si vuole risparmiare occupazione di memoria e tempo di esecuzione - programmare direttamente il microprocessore che lo governa.

La CPU

Il termine CPU è spesso usato con due distinti significati. Considerando il computer completo di periferiche, si indica talvolta con CPU la parte contenente l'unità centrale propriamente detta (quella che esegue le istruzioni nella corretta sequenza), la memoria centrale e le interfacce di collegamento con il resto del calcolatore (e, in molti personal, anche la tastiera). In questo caso CPU (o unità centrale) è sinonimo di: "il computer vero e proprio, escluse le periferiche fisicamente separate (video, registratore, stampante, ecc..)".
Dal punto di vista logico, invece, la CPU è solamente l'unità centrale di calcolo e di controllo, escludendo quindi memorie ed interfacce di qualsiasi tipo. Visto che noi abbiamo sempre inteso il termine "CPU" (ed anche quello di "unità centrale") solo in questo secondo significato, continueremo a farlo anche nel resto dei nostri discorsi.
Fatta questa doverosa precisazione, entriamo subito nel vivo dell'argomento di questa lezione, cioè nella descrizione particolareggiata della CPU.
Finora abbiamo sempre accennato all'unità centrale come a una sorta di "scatola magica", alla quale rivolgevamo in ingresso delle domande (seguendo una determinata sintassi e grammatica) e da cui ottenevamo in uscita delle risposte. Adesso è giunto il momento di rimuovere questa limitazione, cercando di scavare un po' più in profondità. La conoscenza - più o meno approfondita - dell'unità centrale non è solitamente richiesta al programmatore; anzi, nei limiti del possibile, tutti i linguaggi ad alto livello (come il BASIC) cercano di evitare che tra hardware e software esistano legami di interdipendenza.
La fase di scrittura di un certo programma (per lo meno in un linguaggio ad alto livello) dovrebbe infatti essere sempre svolta in assoluta autonomia dalle caratteristiche hardware del computer utilizzato, in modo da risentire il meno possibile delle peculiarità costruttive

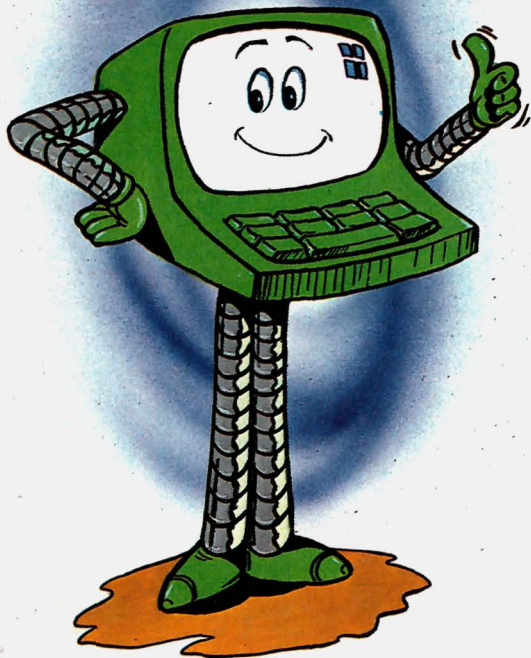
tipiche di un certo sistema.
Il BASIC, come ben sai - e soprattutto a causa di vari motivi storici che ne hanno provocato un'evoluzione anomala -, non può certo essere considerato il linguaggio in assoluto più "trasportabile" e "compatibile" (basta dare un'occhiata a uno stesso programma scritto per due diversi calcolatori per rendersene conto immediatamente!).
Negli ultimi anni sono stati tuttavia sviluppati numerosi altri linguaggi ad alto livello, che permettono di raggiungere l'obbiettivo della portabilità (cioè della quasi perfetta compatibilità tra una macchina e l'altra) in modo praticamente completo.
Tuttavia, in determinate circostanze, soprattutto nei casi di utilizzo particolarmente "evoluto" del computer o di programmazioni in linguaggi a basso livello (cioè molto più vicini alla macchina che all'uomo), può essere invece di estrema utilità fare specifico riferimento a un certo elaboratore e - di conseguenza - anche a uno specifico microprocessore. Oltre che a livello di cultura generale, la conoscenza della costituzione dell'unità centrale - per quanto semplificata e limitata - può pertanto risultare addirittura necessaria o addirittura indispensabile.

Organizzazione esterna di una CPU

Un microprocessore da solo non sa fare niente. Perché possa lavorare occorre collegargli delle memorie e delle interfacce. Una volta collegato a questi circuiti il microprocessore diventa allora un microcalcolatore, generalmente divisibile in quattro parti distinte, che comunicano le une con le altre tramite un bus dati, un bus indirizzi e un bus di controllo (per il momento puoi associare alla parola "bus" il corrispondente italiano di "collegamento"). Queste quattro parti sono:
1) Il microprocessore.
2) La memoria di programma, che segnala alla CPU le funzioni da compiere. Può essere - a seconda dei casi - memoria ROM o memoria RAM.
3) La memoria dati, che immagazzina i dati provenienti dalle periferiche, i risultati intermedi e i risultati finali.
4) Le interfacce, che consentono il trasferimento dei dati dalle periferiche verso il microprocessore e viceversa.
Esaminiamo per un momento più da vicino le ultime tre.
— La memoria di programma contiene la sequenza di istruzioni che il microprocessore deve eseguire. Come già accennato, essa può essere sia di tipo RAM che ROM. Quando per esempio accendi il tuo computer la scritta che compare sullo schermo viene comandata alla CPU da un programma che si trova su ROM, mentre quando scrivi un'istruzione mediante tastiera essa viene memorizzata nella RAM.
— La memoria dati è invece un tipo di memoria notevolmente diverso dalla memoria del programma. È per forza di cose una memoria RAM, visto che deve poter essere letta o scritta tutte le volte che lo desideri. La sua presenza è indispensabile, poiché il microprocessore - per poter eseguire il suo programma - deve utilizzare dei dati che, per forza di cose, sono contenuti nella memoria dati o che provengono dalle periferiche tramite interfacce.
— Le interfacce infine, come ben sai, sono dei circuiti di ingresso-uscita la cui funzione è definita da un programma. Esse consentono la comunicazione tra il microprocessore e le periferiche.

Organizzazione *interna* di una CPU

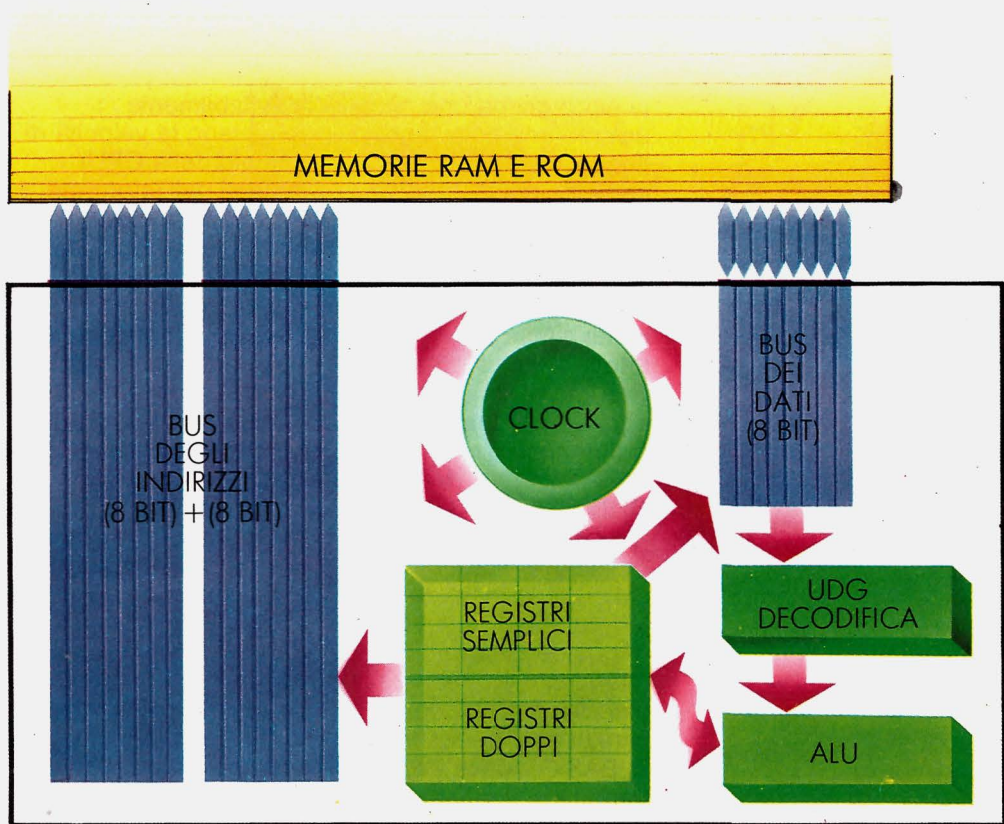
Un microprocessore contiene parecchie migliaia di transistor e di altri componenti elettronici: non è quindi il caso di descriverlo nei minimi particolari (e d'altra parte non ci sarebbe nemmeno di grande utilità).
Un microprocessore è comunque costituito da tre parti principali:
— l'unità di controllo
— l'unità aritmetico-logica
— i registri.
L'unità di controllo decodifica le istruzioni che vengono prelevate dalla memoria di programma ed elabora i segnali di comando necessari all'esecuzione di *un'istruzione*.
La funzione dell'unità aritmetico-logica (brevemente chiamata anche ALU) è invece l'esecuzione delle operazioni logiche ed aritmetiche sui dati che la alimentano attraverso le sue due porte di ingresso, che sono, rispettivamente, "l'ingresso destro" e "*l'ingresso sinistro*": queste porte si possono infatti immaginare come le due estremità più alte di una sagoma a forma di "V".
Dopo l'esecuzione di un'operazione aritmetica, come un'addizione o una sottrazione, l'ALU fa uscire i suoi contenuti dalla punta della "V".
I registri sono di due tipi: *alcuni sono accessibili* dal programma, altri sono interni alla CPU e non hanno un'importanza concettuale rilevante. I registri accessibili si dividono in tre categorie:
— i registri dei dati
— i registri degli indirizzi
— il contatore di programma (Program Counter), il puntatore dello stack e il registro di stato.
I registri dei dati sono in pratica delle locazioni di memoria che consentono la *memorizzazione* temporanea delle informazioni negli spostamenti tra l'unità aritmetico-logica, le memorie e le interfacce.
La loro lunghezza, cioè il numero di bit che compongono ciascun registro, è ovviamente uguale a quella della parola del microprocessore: 8 bit, *se il microprocessore* opera su 8 bit (come per esempio è il caso del 6502, cioè della CPU montata nel VIC 20).
I registri di indirizzo, detti anche puntatori, contengono indirizzi delle posizioni di memoria i quali vengono inviati al bus di indirizzo con un'istruzione particolare che consenta *l'accesso a tali posizioni*.
Ti ricordi quando - parlando delle memorie - accennammo al fatto che ogni locazione disponeva di un ben preciso indirizzo? Bene, questo indirizzo serve proprio ai registri di indirizzo per fare in modo che la CPU possa

riferirsi a qualsiasi posizione della memoria. Il puntatore dello stack è un particolare registro di indirizzo che punta a *una particolare zona* della memoria, chiamata "area di stack". Viene automaticamente decrementato dopo ogni trasferimento di una informazione in questa zona, e incrementato dopo ogni prelievo. Il puntatore di stack, come abbiamo già avuto modo di accennare in una delle scorse lezioni, ha una funzione *particolarmente* importante nella chiamata e nel ritorno dai sottoprogrammi.
Il Program Counter (o contatore delle istruzioni) sorveglia l'esecuzione dell'intero programma. Viene caricato inizialmente con l'indirizzo della prima istruzione del programma, e in seguito segnala al microprocessore gli *indirizzi delle istruzioni* che devono essere eseguite successivamente.
Mentre il microprocessore legge nella memoria di programma una istruzione, e la esegue, il contatore prosegue fino all'indirizzo dell'istruzione successiva e via di seguito. Il microprocessore in genere esegue le istruzioni in ordine sequenziale, cioè una dopo l'altra. Però, nel *caso di alcune istruzioni* l'esecuzione sequenziale dl programma può essere modificata: nel Program Counter viene allora caricato l'indirizzo di salto, e l'eventuale indirizzo di ritorno al programma principale viene conservato per essere utilizzato in seguito.
Il registro di stato contiene invece un certo numero di bit posti a 0 o a 1, a seconda che si siano verificate determinate condizioni dopo l'esecuzione di alcune istruzioni (per esempio, se il risultato è positivo o negativo, con riporto o senza, ecc.). Lo svolgimento del programma può essere modificato in funzione *dei valori assunti da uno* o più bit del registro di stato. È uno dei registri più importanti che il programmatore ha a propria disposizione. I registri possono infatti essere usati per varie verifiche speciali o condizioni eccezionali, oppure per verificare velocemente alcuni risultati errati.
Esiste infine un altro importantissimo registro che equipaggia la ALU: l'accumulatore. Questo è sempre uno dei due ingressi dell'ALU (poco importa se il "destro" o il "sinistro"); l'ALU fa infatti sempre automaticamente riferimento a questo accumulatore come ad uno degli ingressi.

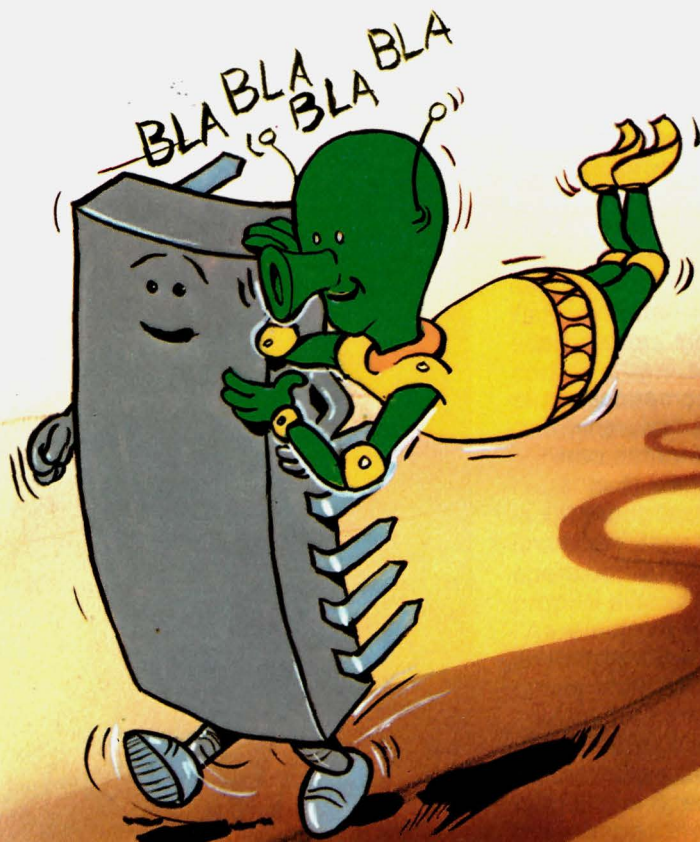
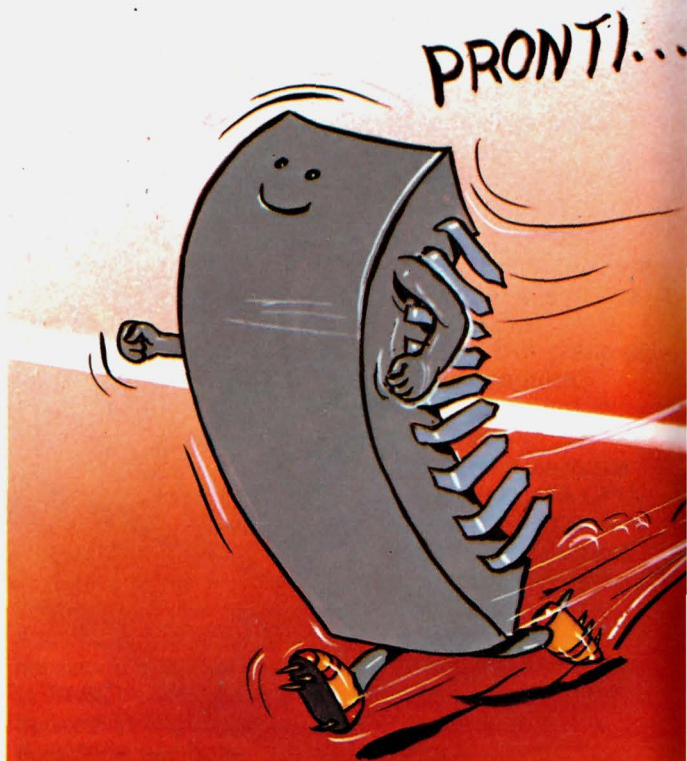
Nelle operazioni aritmetiche e logiche, quindi, uno degli operandi sarà nell'accumulatore e l'altro si troverà tipicamente in una locazione della memoria. Il risultato sarà *depositato* nell'accumulatore. Il riferimento dell'accumulatore come sorgente e destinazione dei dati è la ragione del suo nome: esso accumula i risultati.
Il vantaggio di questo approccio basato sull'accumulatore è costituito dal fatto che si *possono impiegare* istruzioni molto più corte in linguaggio macchina. Se anche l'altro degli operandi dovesse infatti essere prelevato da uno degli altri registri (diversi dall'accumulatore), sarebbe necessario utilizzare istruzioni più lunghe per indicare l'indirizzo del registro. Perciò l'architettura dell'accumulatore si risolve in una maggiore velocità di esecuzione.
Lo svantaggio è che l'accumulatore deve sempre essere caricato *con i dati richiesti* dall'operazione prima della sua utilizzazione. Questo può talvolta provocare qualche punto inefficiente.

I bus

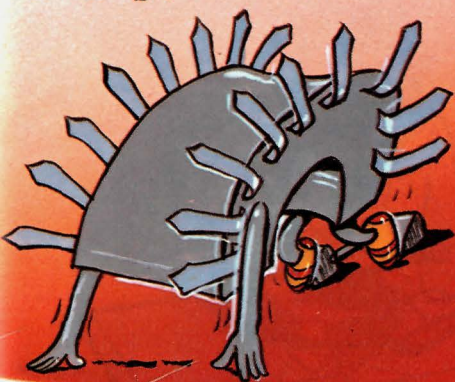
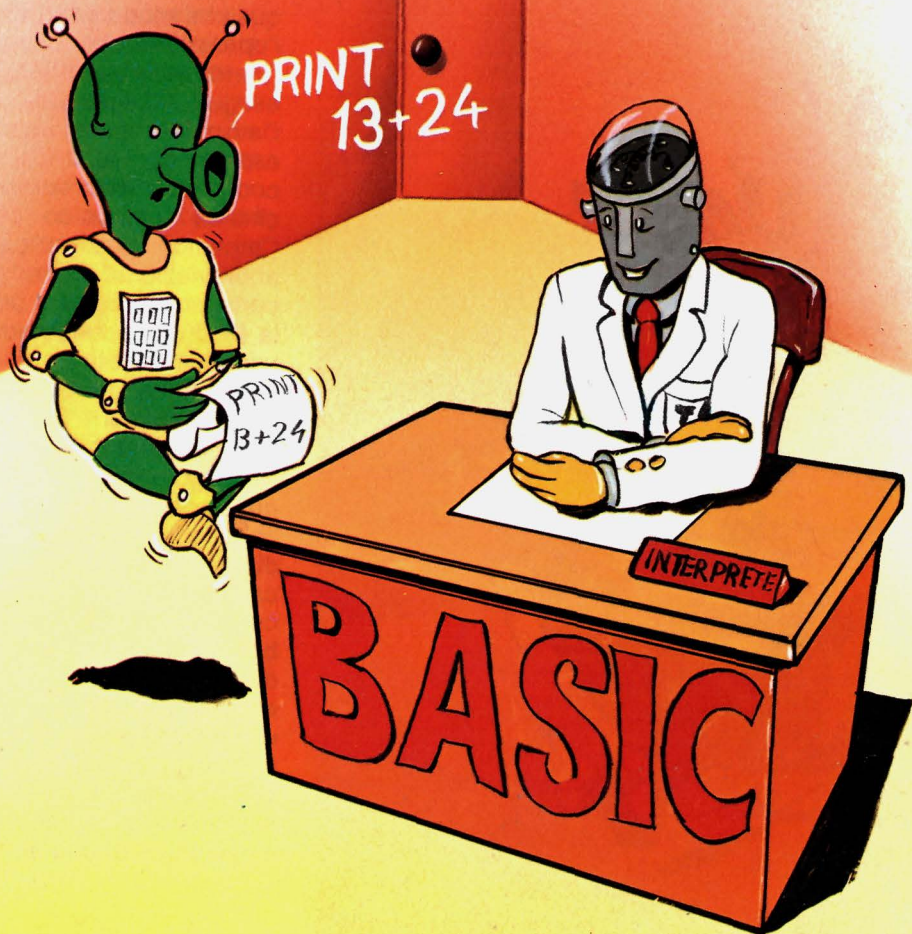
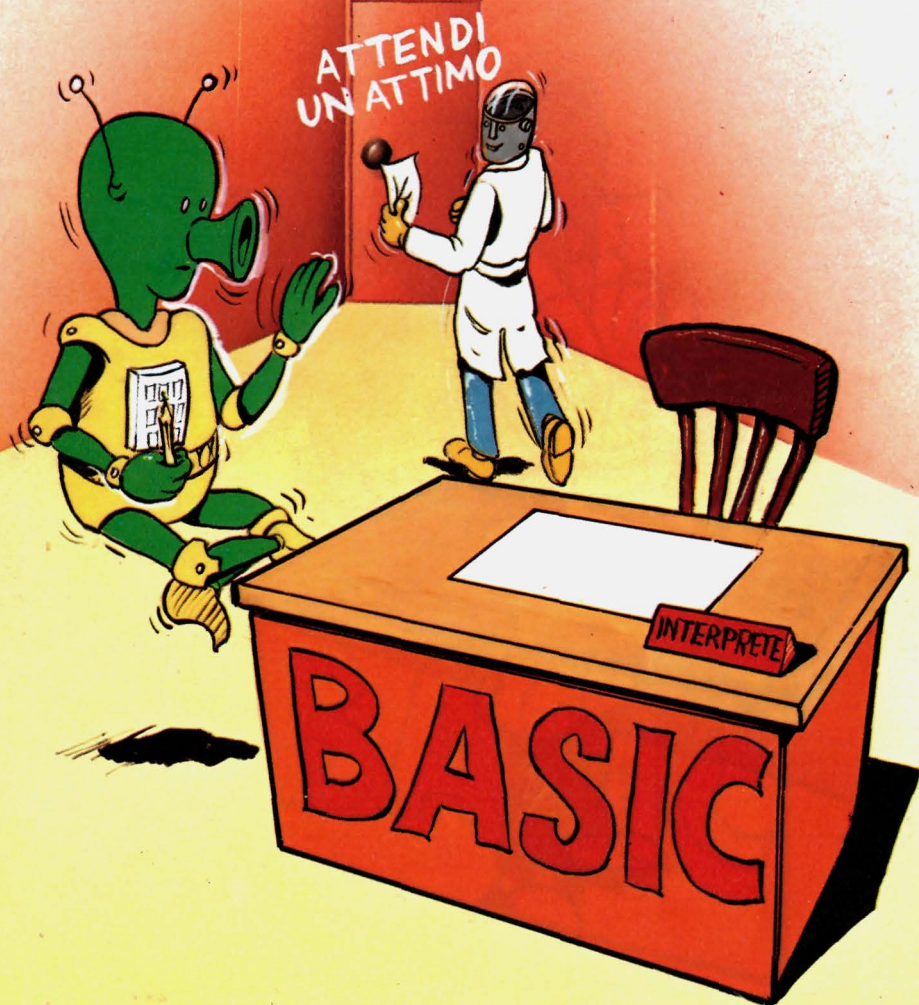
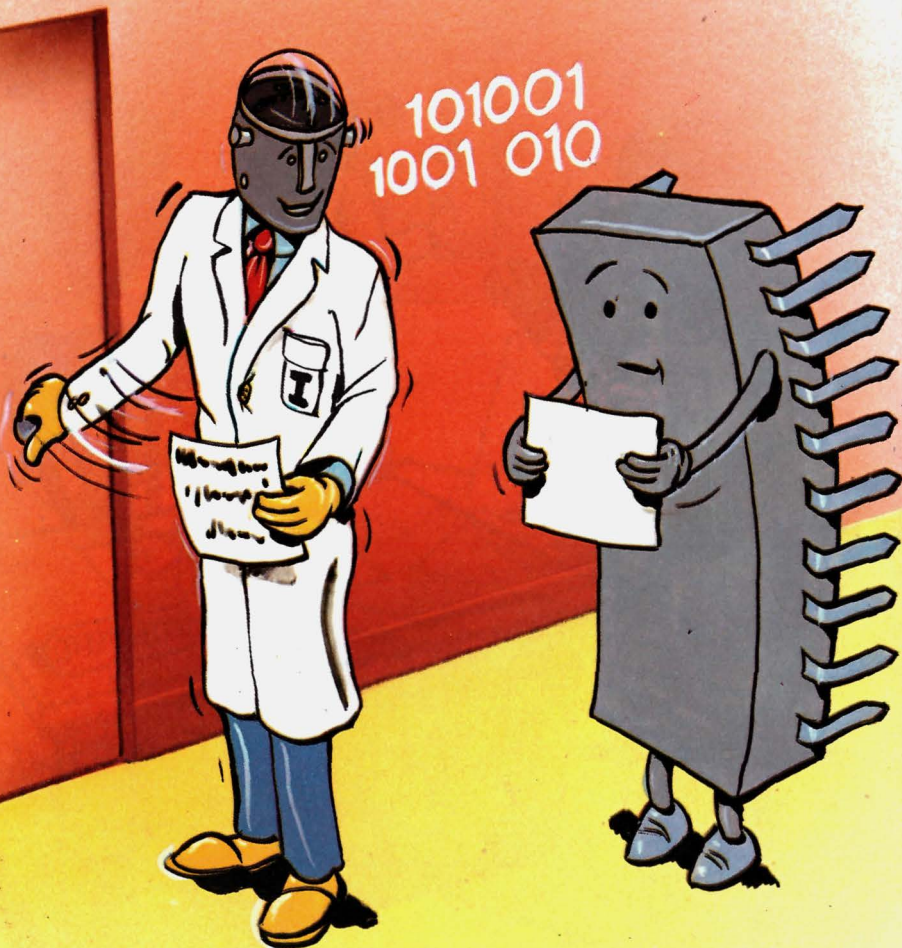
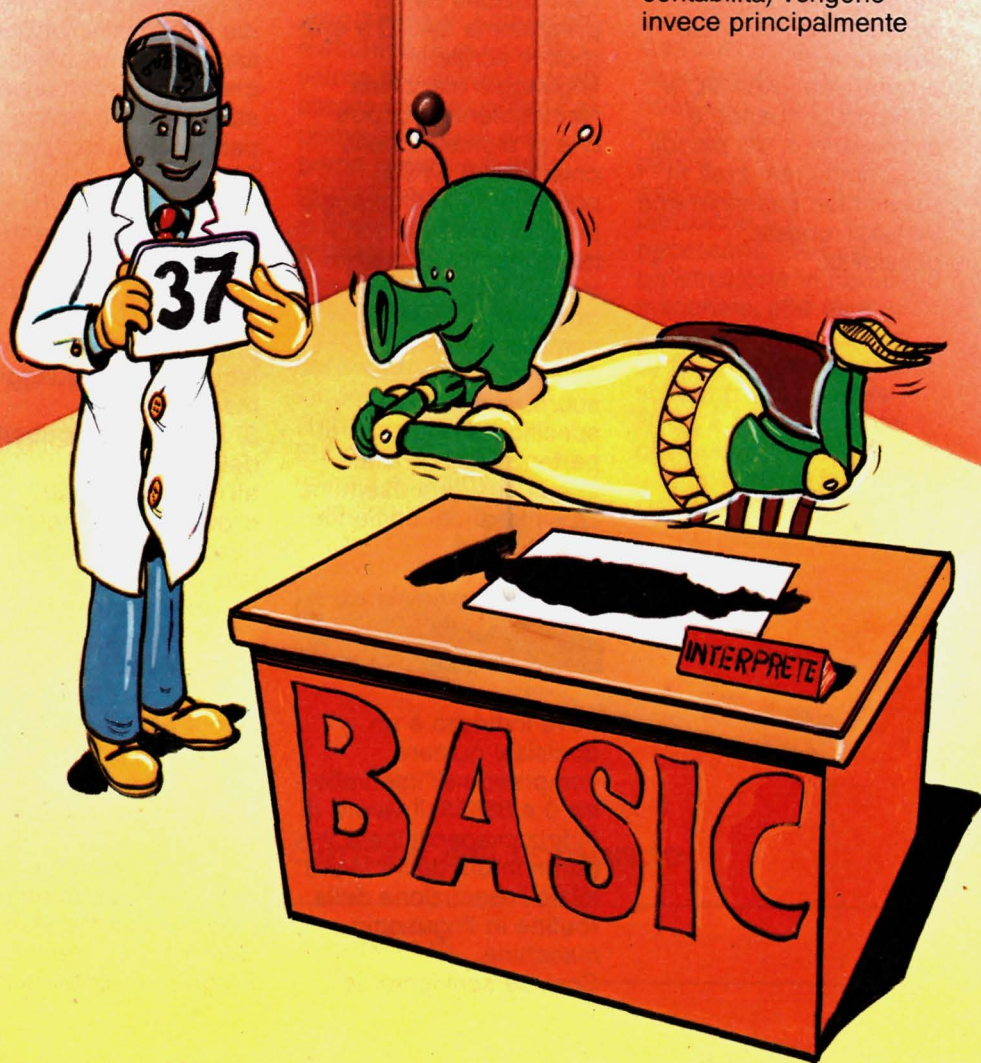
Il microprocessore comunica con le periferiche tramite tre bus:
— il bus degli indirizzi
— il bus dei dati
— il bus di controllo.
Il bus degli indirizzi è un insieme di linee elettriche, che consentono al microprocessore di selezionare una posizione di memoria o un registro di un'interfaccia. La CPU invia su queste linee, verso una periferica, un indirizzo codificato in binario. La periferica, ricevuto e decodificato l'indirizzo, seleziona il registro corrispondente.
Il numero di linee del bus degli indirizzi determina quella che si chiama potenza di indirizzamento del microprocessore; per esempio, 16 linee consentono di indirizzare 2↑16 (65536) locazioni di memoria. È da notare - tra l'altro - che il concetto di indirizzo è molto vicino a quello che si usa nel linguaggio corrente: la localizzazione di una persona in una città avviene in effetti tramite un indirizzo che contiene la strada e il numero civico della strada.

Il bus dei dati è anch'esso costituito da un gruppo di linee elettriche sulle quali avvengono gli scambi di dati tra il microprocessore e le periferiche (memoria ed interfacce). Il numero di linee di questo bus dipende dalla lunghezza della parola del

microprocessore: poiché il microprocessore del VIC 20 è a 8 bit, il bus dati dispone anch'esso di 8 linee.
Il bus di controllo è costituito da un certo numero di segnali di vario genere, che assicurano la sincronizzazione tra il microprocessore e le periferiche. Le funzioni più comuni assicurate da questo bus sono:
— la selezione di un'operazione di lettura o scrittura
— la interruzione del microprocessore (per interruzione si intende una procedura che - attraverso un segnale elettrico a un particolare piedino della CPU - segnala al microprocessore che una periferica vuole comunicare con lui)
— la richiesta di accesso al bus di una periferica
— il riconoscimento di una richiesta di accesso al bus
— altre funzioni meno comuni, ma altrettanto importanti di quelle viste finora.

Il linguaggio macchina

Abbiamo già detto varie volte che scrivere un programma significa scrivere una certa serie di istruzioni in un linguaggio comprensibile dal calcolatore, cioè in un linguaggio di programmazione.
Facciamo per un momento un'analogia con l'uomo. Noi possiamo parlare molte lingue (inglese, francese, tedesco, ecc.), possiamo comprenderle tutte, ma se ci mettiamo a fare un ragionamento o a pensare a un problema, ci accorgiamo che ne usiamo una sola. Questa lingua, usata dal nostro cervello, a volte è la nostra lingua madre - l'italiano -, ma altre volte non è che una "lingua interna". Qualcosa di molto simile accade anche per i calcolatori. Essi possono infatti capire molti linguaggi di programmazione, il BASIC, il FORTRAN, il COBOL, il PASCAL, ecc., ma nel loro interno ne usano uno ed uno soltanto, formato da lunghe file di zeri ed uno in codice binario. Questo linguaggio è quello della macchina. Qualunque istruzione - per esempio in BASIC - deve essere tradotta in linguaggio macchina per poter essere eseguita dal calcolatore. Tale compito nel BASIC del tuo VIC 20 viene realizzato - ormai lo sai benissimo - dall'infaticabile interprete BASIC, vero e proprio traduttore simultaneo tra te e il tuo computer.
In alcuni casi può tuttavia essere utile (o addirittura indispensabile) ricorrere alla programmazione diretta del microprocessore, scavalcando i normali meccanismi (che inevitabilmente rallentano la velocità di calcolo della CPU) imposti dall'interprete.
Programmare in linguaggio macchina significa fornire al computer un certo insieme di configurazioni binarie (chiamate anche codici operativi) che il calcolatore è in grado di capire ed eseguire; ciascuna di esse rappresenta un'operazione eseguita via hardware dalla CPU, cioè mediante delle vere e proprie commutazioni di interruttori all'interno del computer.
La differenza fondamentale è che per programmare in BASIC non è necessario conoscere come

funziona il microprocessore con il quale è realizzato il calcolatore, mentre è necessario per programmare in Assembler (così viene normalmente chiamato il linguaggio macchina). In BASIC, a meno di

usare le istruzioni che leggono e scrivono direttamente nei byte di memoria (cioè PEEK e POKE) non devi mai occuparti di indirizzi di memoria: questo compito è affidato alle routine del sistema operativo e dell'interprete BASIC e viene effettuato in modo completamente automatico.
Una volta che hai imparato il linguaggio BASIC, esso ti mette a disposizione una specie di interfaccia software con la quale tu comunichi e che ti fornisce tutti i mezzi per programmare ed eseguire con successo i *tuoi programmi*.
I dati sono trattati con nomi simbolici (è cioè possibile assegnare un *numero ad ogni nome*: per esempio LET PREZZO = 5000); i riferimenti a punti specifici del programma (GOTO, GOSUB) si ottengono con riferimento ai numeri distintivi delle linee del programma BASIC.
In Assembler, invece, devi occuparti dei registri funzionali del microprocessore, degli indirizzi di memoria e ogni operazione deve essere pensata in tutti i suoi particolari, seguendo le caratteristiche del calcolatore. Inoltre, in Assembler risulta meno semplice il trattamento dei dati.

Svantaggi e vantaggi dell'Assembler

Il linguaggio macchina è in binario: puoi intuire come scrivere un programma in questo linguaggio sia lungo, laborioso e difficile.
D'altra parte i programmi in Assembler consentono una maggiore velocità operativa e un controllo sulla macchina molto più diretto che in BASIC, ed in certi casi è quindi necessario o conveniente sobbarcarsi a questa fatica, quando l'applicazione lo richiede.
Esistono diverse possibilità per scrivere programmi in linguaggio macchina, per esempio utilizzando - anziché un codice numerico - un codice simbolico, in cui ciascuna istruzione viene rappresentata da una parola che in qualche modo ricorda l'operazione che l'istruzione stessa esegue. Ad esempio, l'istruzione di somma si esprime con ADD. Un codice di questo tipo si chiama mnemonico ("mnemonico" significa abbreviazione di una certa istruzione, che per la CPU corrisponde ad una determinata operazione); per essere proprio precisi l'Assembler è il *linguaggio costituito da* questi codici. Dato che il linguaggio comprensibile direttamente dal *calcolatore resta* comunque quello binario, occorre disporre di un programma che effettui la traduzione dall'Assembler al linguaggio macchina vero e proprio (cioè sostituisca al codice mnemonico di una certa

istruzione il corrispondente codice numerico); questo programma prende il nome di assemblatore. Esistono in commercio numerosi assemblatori: il loro utilizzo, per chi desidera programmare seriamente in Assembler, è praticamente indispensabile. Tenuto conto del carattere introduttivo al linguaggio macchina di questa e delle prossime lezioni, non è comunque assolutamente necessario che ti premuri di procurartene uno.
Per concludere, queste sono le principali cause che possono giustificare

l'impiego dell'Assembler:
— tutte le volte in cui il fattore tempo gioca un ruolo fondamentale. Tra il tempo di esecuzione di un programma che funziona sotto l'interprete BASIC e quello di un programma scritto direttamente in Assembler può esserci un'enorme differenza di velocità (il linguaggio macchina può essere fino a 2-300 volte più veloce);
— tutte le volte in cui il fattore spazio è di essenziale importanza. La dimensione di un programma BASIC, aggiunta a quella del suo interprete, sarà sempre superiore a

quella dello stesso programma (che esegue la stessa funzione), realizzato in Assembler. Tale fattore, con il progressivo ed inarrestabile calare dei prezzi delle memorie, sta comunque diventando sempre meno importante;
— tutte le volte in cui è necessario realizzare delle istruzioni sconosciute all'interprete BASIC, o impossibili da realizzare con un linguaggio ad alto livello (come appunto il BASIC).
Dato che un programma in Assembler risulta molto più veloce di un programma BASIC, ma ci vuole molto più tempo a scriverlo e risulta più difficile trovarne gli errori, sarà sempre necessario prendere di volta in volta una

decisione sulla strada che conviene scegliere, valutando bene le diverse esigenze, le possibili alternative ed i rispettivi costi (oltre che in termini di tempo, anche di fatica). Inoltre, devi tener presente che certe particolari applicazioni, come l'utilizzo del computer per giochi di animazione, si possono realizzare bene solo programmando in Assembler. In altri casi (esempio tipico: i programmi di gestione aziendale o di contabilità) vengono invece principalmente

richieste doti di leggibilità e di modificabilità dei programmi, ottenibili esclusivamente utilizzando un linguaggio ad alto livello.
Naturalmente, è sempre possibile (e lo si fa spesso) conciliare le due forme di linguaggio, scrivendo ad esempio l'ossatura del programma in BASIC e ricorrendo, quando è necessario, a dei piccoli programmi realizzati in Assembler. Esamineremo tra poco questa possibilità.

SYS

SYS è un comando del BASIC, abbreviazione della parola "System". Esso risulta molto utile quando si desidera passare direttamente l'esecuzione da un programma BASIC a una routine scritta in linguaggio macchina.
Naturalmente, occorre che al momento della chiamata di SYS il programma in linguaggio macchina sia già stato memorizzato all'interno del tuo VIC 20. È inoltre necessario - proprio come avviene nella chiamata delle subroutine in BASIC - specificare l'indirizzo di partenza della routine che si desidera eseguire.
La sintassi del comando è

```
SYS indirizzo
```

dove indirizzo è un qualsiasi numero compreso nell'intervallo tra 0 e 65535. Il valore di indirizzo specificherà il punto in cui dovrà avere inizio l'esecuzione della routine in linguaggio macchina.
Così, se scriviamo un programma contenente una linea BASIC del tipo:

```
100 SYS 2100
```

il nostro VIC 20 eseguirà, nell'ordine, questa serie di operazioni:
1) Passerà il controllo dal BASIC alla routine in linguaggio macchina, che in precedenza sarà stata memorizzata a partire dalla locazione 2100.
2) Eseguirà e porterà a termine (naturalmente quando questa eventualità sarà stata prevista dal programmatore) le varie istruzioni in linguaggio macchina.
3) Ritornerà il controllo della macchina all'interprete BASIC, che provvederà a continuare l'esecuzione delle istruzioni BASIC successive alla linea 100.
Adesso che a grandi linee hai capito come lavora SYS, possiamo entrare un po' più nei dettagli sul funzionamento di questa istruzione. Al momento dell'esecuzione del comando l'elaboratore memorizza sullo stack il valore attuale del Program Counter (ti

ricordi? è il registro della CPU che ricorda al microprocessore gli indirizzi delle varie istruzioni ancora da eseguire): in questo modo al termine della routine in linguaggio macchina l'interprete BASIC saprà ritrovare - prelevandone l'indirizzo dallo stack - la linea di programma da cui continuare l'esecuzione.
Nel Program Counter viene quindi posto l'indirizzo specificato nella SYS e da quell'indirizzo partirà il programma in linguaggio macchina. È ovvio che il valore dell'indirizzo deve essere corrispondente al primo byte (cioè alla prima istruzione) della routine da eseguire.
A questo punto la CPU prosegue nell'esecuzione del programma in linguaggio macchina: se ad un certo istante essa incontra un comando RTS (ReTurn from Subroutine), verrà avviata ed infine completata la procedura per il ritorno al BASIC: il valore precedente del PC verrà tolto dallo stack e l'esecuzione proseguirà normalmente sotto la guida dell'interprete.
Per quanto gli esempi esaminati più avanti (nella parte della lezione dedicata alla programmazione) chiariscano questo fatto, è importante ricordare che con SYS il computer esegue una vera e propria chiamata a una subroutine (in questo caso scritta - lo ripetiamo - in linguaggio macchina). In altre parole, dopo aver fatto eseguire al VIC 20 una istruzione SYS.... è necessario inserire un'istruzione in codice macchina, che dica al computer di tornare al BASIC (questa istruzione abbiamo visto essere RTS). È la stessa cosa che succede quando si usa un'istruzione GOSUB; per tornare al programma principale occorre inserire al termine del sottoprogramma un comando RETURN.

Sintassi del comando

SYS indirizzo

USR

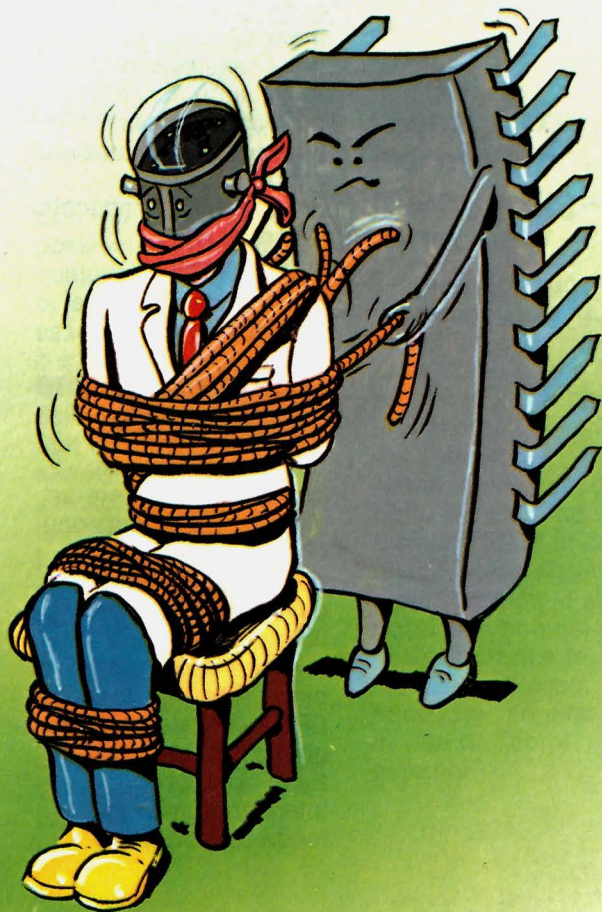
USR (abbreviazione di User SubRoutine) è un'altra istruzione - per precisione è una funzione - utilizzabile per eseguire da BASIC chiamate a parti di programmi scritti in linguaggio macchina.
Essa esegue all'incirca lo stesso compito svolto da SYS, *ma lo fa in* modo leggermente più complicato. Per questa ragione USR viene usata meno frequentemente di SYS.
La sintassi di questa funzione è

```
A = USR (C)
```

e sta a significare per il computer: «trasferisci il valore della variabile C al programma in linguaggio macchina il cui indirizzo di partenza è memorizzato nelle locazioni numero 1 e 2. All'uscita dalla routine assegna il risultato alla variabile A».
Parlando in termini un poco più rigorosi (ma sicuramente più esatti e precisi) USR serve per eseguire un programma in linguaggio macchina, il cui indirizzo deve essere stato preventivamente memorizzato nei byte 1 e 2, con delle istruzioni POKE. Il valore dell'argomento di USR viene memorizzato in una parte del computer chiamata "accumulatore floating-point", che inizia al byte 61; in tale accumulatore (paragonabile né più né meno a una cella di memoria) si trova poi il *risultato del calcolo* eseguito. Il risultato viene comunque reso disponibile al programma BASIC, come valore della funzione.
Anche in questo caso la routine in linguaggio macchina deve terminare con l'istruzione RTS.
Il vantaggio di URS rispetto a SYS è che lo stesso programma può essere usato per *lavorare più volte con* diversi valori in ingresso.
Lo svantaggio è invece costituito dal fatto che è possibile passare alla routine un solo valore alla volta. Inoltre, utilizzando più di una routine è necessario - prima di accedere alla routine - inserire con delle POKE nelle locazioni 1 e 2 l'indirizzo

di partenza del programma. Nei casi in cui il fattore velocità risulti essenziale questo fatto può essere estremamente svantaggioso, richiedendo - tutto sommato - un'inutile perdita di tempo.

Sintassi della funzione

USR (argomento)

Alla scoperta dell'Assembler

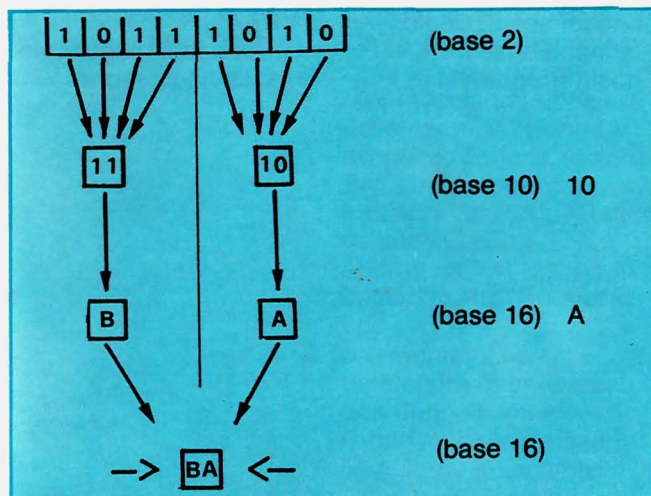
Prima di passare direttamente alla scrittura di programmi in linguaggio macchina dobbiamo ancora affrontare importanti (anzi, indispensabili) argomenti, grazie ai quali potremo successivamente avanzare nei nostri discorsi in modo più semplice e spedito.
Ti potrà forse sembrare che il linguaggio macchina non faccia per te: troppe cose da imparare, da sapere e da tenere a mente.
Questo può anche essere vero: tuttavia, una volta imparati i concetti fondamentali molti argomenti ti diventeranno molto meno complicati di quello che potevano apparire a prima vista.
Inoltre, più ti addentrerai nell'Assembler - diventando di conseguenza più padrone della situazione - più i risultati diventeranno spettacolari e carichi di soddisfazioni.

La numerazione esadecimale

Il primo, importante punto è determinare un sistema di numerazione che giunga a un compromesso tra la difficoltà che l'uomo incontra nel leggere e lavorare con i numeri binari e l'avversità che la CPU possiede verso le cifre decimali.
Ecco quindi che interviene un terzo e fondamentale (per chi programma in Assembler) sistema di numerazione, cioè quello in base 16. Con questa base un numero binario a otto cifre può essere scritto con due cifre in base sedici, cioè con due numeri esadecimali.
Ricorderai certamente che un numero in base due è composto solo da cifre 0 e 1, mentre uno in base 10 è scritto con cifre comprese tra 0 e 9; così un numero in base 16 dovrà essere scritto con cifre comprese tra 0 e 15. Però, dal momento che non abbiamo cifre maggiori di 9, usiamo le prime sei lettere dell'alfabeto:

10 = A
11 = B
12 = C
13 = D
14 = E
15 = F

Il modo più semplice di convertire un numero binario a 8 bit in un numero esadecimale è quello di separare gli 8 bit in due gruppi di 4 bit, effettuare la conversione da una base all'altra di ciascun gruppo e poi ricombinare il risultato.

Per esempio:

Con questo metodo possiamo convertire qualunque valore di un byte in due cifre esadecimali. *Il vantaggio* è quello di avere una scrittura più compatta della notazione binaria e, con un minimo di allenamento, quasi identica al sistema decimale. In ogni caso, tutti questi sistemi di numerazione sono - ricordiamolo - modi diversi di rappresentare gli stessi numeri. Tutto sta nel conoscere la base utilizzata. Per convenzione, allo scopo di riconoscere immediatamente un numero esadecimale, scriveremo sempre i numeri in base 16 con il prefisso $ così

```
10
```

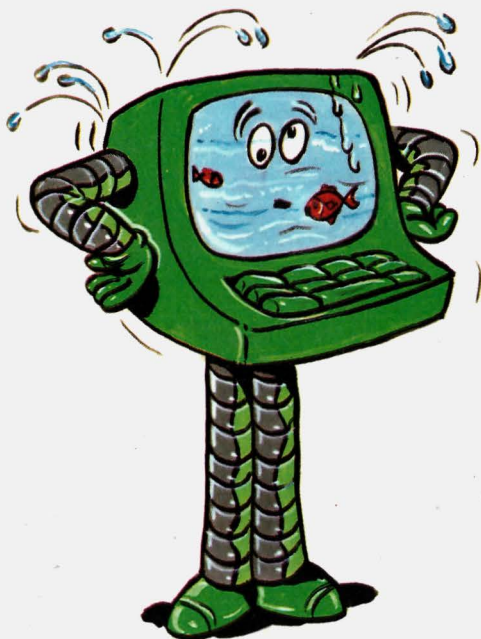
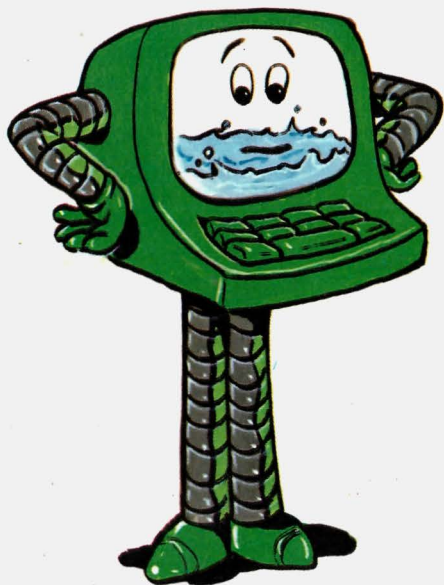
significherà "10 decimale" mentre

```
$ 10
```

significherà "uno zero esadecimale" (cioè 16 decimale).

Come memorizzare i programmi in linguaggio macchina

Una volta che un programma in linguaggio macchina è stato scritto resta come ultimo problema il compito di inserirlo nella memoria del computer. Naturalmente, occorre porlo in una zona della memoria RAM in cui niente possa disturbarlo, cioè dove non vi siano possibilità che nel corso dell'esecuzione le locazioni da esso occupate vengano invase da "corpi estranei".

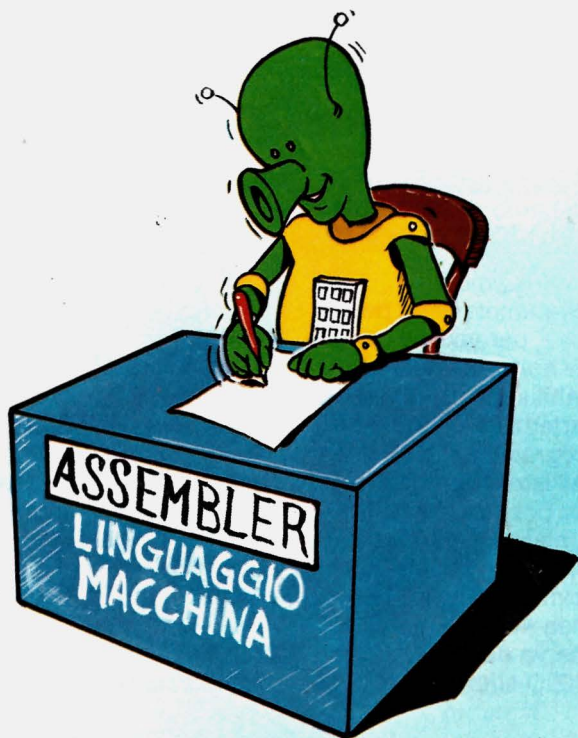
Un ottimo posto, per esempio, è quello che parte dalla locazione 49152 e arriva alla 53247: in questa area di memoria esiste molto spazio inutilizzato, che sembra fatto apposta per i nostri scopi. Decidiamo quindi di inserire tutti i nostri programmi in linguaggio macchina a partire dalla locazione 49152.
Le routine in linguaggio macchina possono

essere incorporate nei programmi BASIC, usando le istruzioni DATA seguite da una serie di numeri che rappresentano i codici delle varie istruzioni da eseguire. Poi una routine BASIC provvederà a trasferire i valori dei byte nella zona di memoria opportuna, utilizzando il comando POKE. Questo che segue è un programma che ti permetterà di caricare, una dopo l'altra, le istruzioni delle routine in linguaggio macchina:

```
10 LET INIZIO = 49152
20 LET FINE = 53247
30 RESTORE
40 FOR A = INIZIO TO FINE
50 READ X : IF X = 999 THEN END
60 POKE A, X
70 NEXT A
80 DATA ...qui vanno scritti i codici...
```

Esaminiamo il programma linea per linea, per vedere come funziona: le linee 10 e 20 stabiliscono l'indirizzo di partenza e di fine del programma. La 30 fa uso dell'istruzione RESTORE, per posizionare il puntatore DATA sul primo dei dati che dovranno essere letti. La linea 40 inizia un ciclo che va dalla locazione INIZIO sino alla FINE. La linea 50 legge i vari elementi delle righe DATA e controlla man mano se ve n'è uno pari a 999 (che è un codice non ammesso per l'istruzione POKE); in questo modo siamo in grado di identificare la fine dei dati.
La linea 70 chiude il ciclo. Naturalmente, non è necessario avere tutti i valori su un'unica linea DATA; puoi usare quante linee desideri, compatibilmente con la memoria disponibile. La linea 80 (e quelle eventualmente seguenti) contiene i valori decimali della routine e deve avere come ultimo termine il 999, affinché il programma possa accorgersi di essere arrivato in fondo.
Così questo programma carica il codice macchina nella RAM.
Per far eseguire la routine occorre modificare il programma, in modo che al termine esso esegua la routine, includendo quindi una linea SYS o USR. Questo esercizio (tutto sommato veramente elementare) lo lasciamo fare a te. In alternativa puoi comandare l'esecuzione impartendo uno dei due comandi in modo immediato.

Esempi Assembler

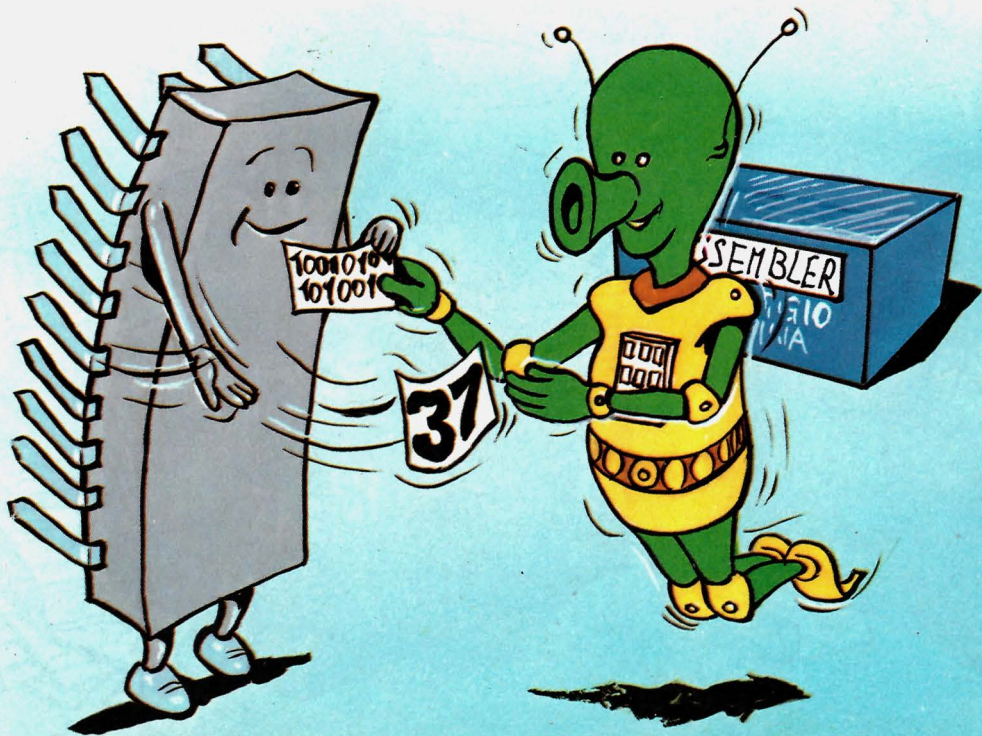
Proviamo adesso a scrivere qualche programma molto semplice in Assembler, illustrandone contemporaneamente il funzionamento e il confronto con il corrispondente listato BASIC. Non sempre è possibile fare un tale paragone: gli esempi che tratteremo ci daranno tuttavia questa possibilità.

Come primo esempio proponiamoci di trasferire il contenuto di due locazioni di memoria (per esempio la 58000 e la 58001) in un'altra coppia di locazioni (scegliamo la 34567 e la 34568). In BASIC scriveremmo:

```
10 POKE 34567, PEEK (58000)
20 POKE 34568, PEEK (58001)
```

e tutto sarebbe risolto. Vediamo adesso come fare in Assembler. Ecco qui il listato:

```
LDA $E290
STA $8707
LDA $E291
STA $8708
RTS
```

La prima riga carica nell'accumulatore (LDA è abbreviazione di LoaD Accumulator, carica l'accumulatore) il valore contenuto nella locazione 58000 (E290 esadecimale). La seconda riga memorizza invece tale valore (STA significa STore Accumulator, memorizza l'accumulatore) nella locazione 34567 (8707

esadecimale). Lo stesso compito (eseguito però sulle celle 58001 e 34568) viene portato a termine sulle due linee successive. L'ultima riga è il famoso RTS (ReTurn from Subroutine), che serve per restituire il controllo al BASIC.
Notiamo ora la costituzione delle istruzioni: un operatore e uno o più operandi. In LDA $E290 - per esempio - LDA è l'operatore, mentre $E290 è l'operando. È necessario fare una precisazione sulla parte (o, meglio, sul campo) relativa agli operandi. Questo campo dipende strettamente dal tipo di operazione, ma è sempre e comunque appartenente a uno dei seguenti gruppi:
— a un solo argomento
— a due argomenti, separati da un carattere speciale (generalmente la virgola).
Passiamo ora alla conversione del programma nei codici numerici (questi che abbiamo appena visto sono solo i mnemonici). Occorre prendere in mano la tabella delle istruzioni, fornita dal costruttore del microprocessore, e convertire uno ad uno i vari termini. A ciascun elemento corrisponderà naturalmente un preciso codice numerico:

LDA $E290	AD 90 E2
STA $8707	8D 07 87
LDA $E291	AD 91 E2
STA $8708	8D 08 87
RTS	60

I numeri che abbiamo scritto sono ancora in esadecimale: prima di essere scritti nelle DATA dovranno essere convertiti in decimali.
L'ultima cosa da notare è che - se guardi bene - gli indirizzi delle locazioni sono stati scritti ribaltati (cioè E290 è stato scritto 90E2, e analogamente per gli altri). Ciò è dovuto a ragioni costruttive del microprocessore, il quale pretende che di qualsiasi indirizzo gli vengano forniti prima il byte basso e poi quello alto.
Alla fine otteniamo i seguenti valori:

$AD	173
$90	144
$E2	226
$8D	141
$07	7
$87	135
$AD	173
$91	145
$E2	226
$8D	141
$08	8
$87	135
$60	96

che, introdotti nella linea DATA (ricordandosi di aggiungere anche il 999), formeranno la routine.
Esegui il programma e comanda quindi la SYS per lanciare la routine in linguaggio macchina.
Successivamente prova a verificare - con delle PEEK nelle locazioni 34567, 34568, 58000, 58001 - che effettivamente il lavoro sia stato portato a termine.
Proviamo ora questo secondo esempio: desideriamo porre nella locazione di memoria numero 20000 il valore 19. Successivamente vogliamo che questo numero venga raddoppiato e sia messo nella locazione 20001.

In BASIC scriveremmo:

```
10 POKE 20000, 19
20 POKE 20001, PEEK (20000) * 2
```

e l'interprete si occuperebbe delle varie operazioni. In Assembler dobbiamo invece pensare noi a tutto. Ecco il listato:

```
LDA $13
STA $4E20
CLC
ROL A
STA $4E21
RTS
```

Cerchiamo di capire cosa è stato fatto: la prima riga carica nell'accumulatore il valore esadecimale 13 (19 decimale), mentre la seconda lo memorizza nella locazione $4E20 (20000 decimale). A questo punto ci si prepara per la moltiplicazione. Poiché in binario moltiplicare per 2 significa spostare di una posizione verso sinistra ciascun bit (per esempio 100 binario - pari a 4 - diventa 1000 - pari a 8 -), basterà eseguire una cosiddetta operazione di rotazione (mnemonico ROL).
Poiché il bit che viene aggiunto sulla destra del numero viene preso dal carry (che è un particolare bit facente parte del registro di stato del microprocessore), bisognerà porlo a zero, per evitare che influenzi erroneamente il risultato: CLC (CLean the Carry, azzera il carry) serve proprio a questo.
Il risultato viene infine memorizzato in $4E21 (20001 decimale).
La conversione del programma nei codici numerici porta ai seguenti valori:

```
169, 19, 141, 32, 78, 24, 42, 141, 33, 78, 96, 999
```

Se eseguirai questa routine, potrai verificare che i due programmi - BASIC e linguaggio macchina - portano a compimento lo stesso lavoro e giungono al medesimo risultato.

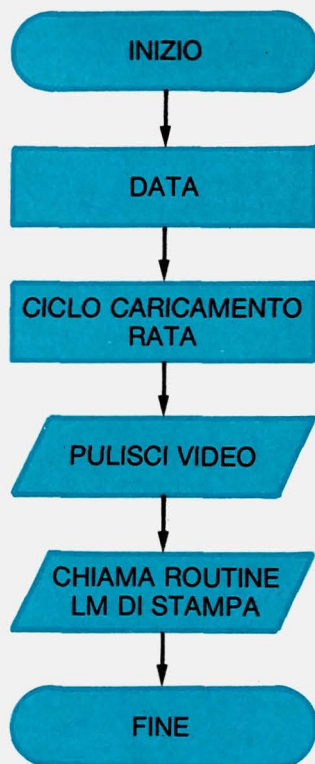
Stampa di un carattere in linguaggio macchina

Al di là della reale possibilità di utilizzo, l'obiettivo importante di questa proposta è quello di sottolineare il *corretto* procedimento da seguire per impostare ed eseguire un qualsiasi programma in linguaggio macchina.
Per stampare infatti in alto a sinistra sullo schermo una pallina, è certamente più pratico utilizzare una linea BASIC del tipo

```
10 PRINT "■ ●"
```

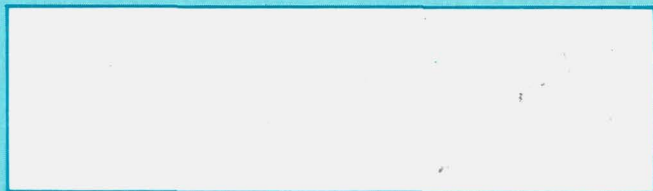
consoliamoci però, per il momento, sapendo che l'operazione di stampa viene eseguita circa 200 volte più velocemente.

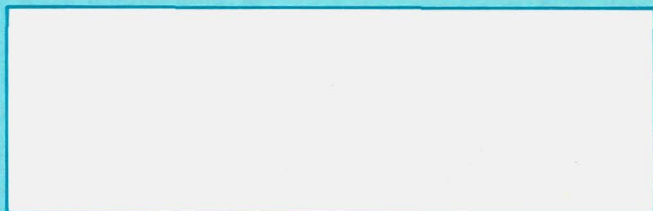
Questo è il disassemblato delle routine introdotta nelle linee 20-25-30 ed eseguita in linea 35.

033C	A9 51	LDA #$51
033E	8D 00 1E	STA $1E00
0341	A9 02	LDA #$02
0343	8D 00 96	STA $9600
0346	60	RTS
0347	00	BRK

```
10 DATA 169, 81, 141, 0, 30
15 DATA 169, 2, 141, 0, 150, 96
20 FOR C = 0 TO 10
25 READ A : POKE 828 + C, A
30 NEXT C
35 PRINT "■" : SYS 828
40 END
```

VIDEOESERCIZI

Cerca di prevedere cosa apparirà sullo schermo dopo l'esecuzione del seguente programma.

```
10 FOR I = 1 TO .002E3
20 PRINT I : NEXT
30 SYS 64824
40 PRINT "DI QUI NON PASSA DI SICURO"
```

Questa breve routine in linguaggio macchina introdotta direttamente in memoria coi POKE, puoi usarla nei tuoi programmi per stupire gli amici.

```
10 POKE 251, 238 : POKE 252, 15 : POKE 253, 144 : POKE 254, 96
20 SYS 251
30 GET A$ : IF A$ = " " THEN GOTO 20
40 REM PREMI UN TASTO PER INTERROMPERE
50 END
```

La routine attivata dalla linea 20 riguarda la stampante, il registratore o il disco?

```
10 PRINT "♥"
20 SYS 63686
30 REM SCOPRI L'ARCANO
```